Exhibit T-4

Definition

```
def AddressImmediate(op,rd,rc,imm) as
    i ← $\text{imm}_{11}^{52}$ || imm
    c ← RegRead(rc, 64)
    case op of
        A.SUB.I:
            a ← i - c
        A.SUB.I.O:
            t ← ($i_{63}$ || i) - ($c_{63}$ || c)
            if $t_{64} \neq t_{63}$ then
                raise FixedPointArithmetic
            endif
            a ← $t_{63..0}$
        A.SUB.I.U.O:
            t ← ($i_{63}$ || i) - ($c_{63}$ || c)
            if $t_{64} \neq 0$ then
                raise FixedPointArithmetic
            endif
            a ← $t_{63..0}$
        A.SET.AND.E.I:
            a ← $((i \text{ and } c) = 0)^{64}$
        A.SET.AND.NE.I:
            a ← $((i \text{ and } c) \neq 0)^{64}$
        A.SET.E.I:
            a ← $(i = c)^{64}$
        A.SET.NE.I:
            a ← $(i \neq c)^{64}$
        A.SET.L.I:
            a ← $(i < c)^{64}$
        A.SET.GE.I:
            a ← $(i \geq c)^{64}$
        A.SET.L.I.U:
            a ← $((0 \| i) < (0 \| c))^{64}$
        A.SET.GE.I.U:
            a ← $((0 \| i) \geq (0 \| c))^{64}$
    endcase
    RegWrite(rd, 64, a)
enddef
```

Exceptions

Fixed-point arithmetic

FIG. 63C

Operation codes

A.SET.AND.E	Address set and equal zero
A.SET.AND.NE	Address set and not equal zero
A.SET.E	Address set equal
A.SET.GE	Address set greater equal signed
A.SET.GE.U	Address set greater equal unsigned
A.SET.L	Address set less signed
A.SET.L.U	Address set less unsigned
A.SET.NE	Address set not equal
A.SUB	Address subtract
A.SUB.O	Address subtract signed check overflow
A.SUB.U.O	Address subtract unsigned check overflow

Equivalencies

A.SET.E.Z	Address set equal zero
A.SET.G.Z	Address set greater zero signed
A.SET.GE.Z	Address set greater equal zero signed
A.SET.L.Z	Address set less zero signed
A.SET.LE.Z	Address set less equal zero signed
A.SET.NE.Z	Address set not equal zero
A.SET.G	Address set greater signed
A.SET.G.U	Address set greater unsigned
A.SET.LE	Address set less equal signed
A.SET.LE.U	Address set less equal unsigned

A.SET.E.Z rd=rc	←	A.SET.AND.E rd=rc,rc
A.SET.G.Z rd=rc	⇐	A.SET.L.U rd=rc,rc
A.SET.GE.Z rd=rc	⇐	A.SET.GE rd=rc,rc
A.SET.L.Z rd=rc	⇐	A.SET.L rd=rc,rc
A.SET.LE.Z rd=rc	⇐	A.SET.GE.U rd=rc,rc
A.SET.NE.Z rd=rc	←	A.SET.AND.NE rd=rc,rc
A.SET.G rd=rb,rc	→	A.SET.L rd=rc,rb
A.SET.G.U rd=rb,rc	→	A.SET.L.U rd=rc,rb
A.SET.LE rd=rb,rc	→	A.SET.GE rd=rc,rb
A.SET.LE.U rd=rb,rc	→	A.SET.GE.U rd=rc,rb

Redundancies

A.SET.E rd=rc,rc	⇔	*A.SET rd*
A.SET.NE rd=rc,rc	⇔	*A.ZERO rd*

FIG. 64A

Selection

class	operation	cond	operand	check
arithmetic	SUB			
			NONE U	O
boolean	SET.AND SET	E NE		
	SET	L GE *G LE*	NONE U	
	SET	*L GE G LE E NE*	Z	

Format

op rd=rb,rc

rd=op(rb,rc)
rd=opz(rcb)

31 24	23 18	17 12	11 6	5 0
A.MINOR	rd	rc	rb	op
8	6	6	6	6

rc ← rb ← rcb

FIG. 64B

Definition

```
def AddressReversed(op,rd,rc,rb) as
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case op of
        A.SET.E:
            $a \leftarrow (b = c)^{64}$
        A.SET.NE:
            $a \leftarrow (b \neq c)^{64}$
        A.SET.AND.E:
            $a \leftarrow ((b \text{ and } c) = 0)^{64}$
        A.SET.AND.NE:
            $a \leftarrow ((b \text{ and } c) \neq 0)^{64}$
        A.SET.L:
            $a \leftarrow ((rc = rb)\ ?\ (b < 0) : (b < c))^{64}$
        A.SET.GE:
            $a \leftarrow ((rc = rb)\ ?\ (b \geq 0) : (b \geq c))^{64}$
        A.SET.L.U:
            $a \leftarrow ((rc = rb)\ ?\ (b > 0) : ((0 \| b) < (0 \| c))^{64}$
        A.SET.GE.U:
            $a \leftarrow ((rc = rb)\ ?\ (b \leq 0) : ((0 \| b) \geq (0 \| c))^{64}$
        A.SUB:
            $a \leftarrow b - c$
        A.SUB.O:
            $t \leftarrow (b_{63} \| b) - (c_{63} \| c)$
            if $t_{64} \neq t_{63}$ then
                raise FixedPointArithmetic
            endif
            $a \leftarrow t_{63..0}$
        A.SUB.U.O:
            $t \leftarrow (0^1 \| b) - (0^1 \| c)$
            if $t_{64} \neq 0$ then
                raise FixedPointArithmetic
            endif
            $a \leftarrow t_{63..0}$
    endcase
    RegWrite(rd, 64, a)
enddef
```

Exceptions

Fixed-point arithmetic

FIG. 64C

Operation codes

A.SHL.I.ADD	Address shift left immediate add

FIG. 65A

Format

A.SHL.I.ADD rd=rc,rb,i

rc=op(ra,rb,i)

31 24	23 18	17 12	11 6	5 2	1 0
A.MINOR	rd	rc	rb	ASHL.I.ADD	sh
8	6	6	6	6	2

assert 1≤i≤4
sh ← i-1

FIG. 65B

Definition

```
def AddressShiftLeftImmediateAdd(sh,rd,rc,rb) as
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 64)
    a ← c + ($b_{62-sh..0}$ || $0^{1+sh}$)
    RegWrite(rd, 64, a)
enddef
```

Exceptions

none

FIG. 65C

Operation codes

A.SHL.I.SUB	Address shift left immediate subtract

FIG. 66A

Format

ASHL.I.SUB rd=rb,i,rc

rd=op(rb,i,rc)

31 24	23 18	17 12	11 6	5 2	1 0
A.MINOR	rd	rc	rb	ASHL.I.SUB	sh
8	6	6	6	6	2

assert $1 \leq i \leq 4$
sh ← i-1

FIG. 66B

Definition

```
def AddressShiftLeftImmediateSubtract(op,rd,rc,rb) as
        $c \leftarrow$ RegRead(rc, 128)
        $b \leftarrow$ RegRead(rb, 128)
        $a \leftarrow (b_{62\text{-}sh..0} \| 0^{1+sh}) - c$
        RegWrite(rd, 64, a)
enddef
```

Exceptions

none

FIG. 66C

Operation codes

A.SHL.I	Address shift left immediate
A.SHL.I.O	Address shift left imMediate signed check overflow
A.SHL.I.U.O	Address shift left immediate unsigned check overflow
A.SHR.I	Address signed shift right immediate
A.SHR.I.U	Address shift right immediate unsigned

Redundancies

A.SHL.I rd=rc,1	⇔	A.ADD rd=rc,rc
A.SHL.I.O rd=rc,1	⇔	A.ADD.O rd=rc,rc
A.SHL.I.U.O rd=rc,1	⇔	A.ADD.U.O rd=rc,rc
A.SHL.I rd=rc,0	⇔	*A.COPY rd=rc*
A.SHL.I.O rd=rc,0	⇔	*A.COPY rd=rc*
A.SHL.I.U.O rd=rc,0	⇔	*A.COPY rd=rc*
A.SHR.I rd=rc,0	⇔	*A.COPY rd=rc*
A.SHR.I.U rd=rc,0	⇔	*A.COPY rd=rc*

FIG. 67A

Selection

class	operation	form	operand	check
shift	SHL	I		
			NONE U	O
	SHR	I	NONE U	

Format

op rd=rc,simm

rd=op(rc,simm)

31 24	23 18	17 12	11 6	5 0
A.MINOR	rd	rc	simm	op
8	6	6	6	6

FIG. 67B

Definition

```
def AddressShiftImmediate(op,rd,rc,simm) as
    c ← RegRead(rc, 64)
    case op of
        A.SHL.I:
            $a \leftarrow c_{63-simm..0} \| 0^{simm}$
        A.SHL.I.O:
            if $c_{63..63-simm} \neq c_{63}^{simm+1}$ then
                raise FixedPointArithmetic
            endif
            $a \leftarrow c_{63-simm..0} \| 0^{simm}$
        A.SHL.I.U.O:
            if $c_{63..64-simm} \neq 0$ then
                raise FixedPointArithmetic
            endif
            $a \leftarrow c_{63-simm..0} \| 0^{simm}$
        A.SHR.I:
            $a \leftarrow a_{63}^{simm} \| c_{63..simm}$
        A.SHR.I.U:
            $a \leftarrow 0^{simm} \| c_{63..simm}$
    endcase
    RegWrite(rd, 64, a)
enddef
```

Exceptions

Fixed-point arithmetic

FIG. 67C

Operation codes

A.MUX	Address multiplex

FIG. 68A

Format

op ra=rd,rc,rb

ra=amux(rd,rc,rb)

31 24	23 18	17 12	11 6	5 0
op	rd	rc	rb	ra
8	6	6	6	6

FIG. 68B

Definition

```
def AddressTernary(op,rd,rc,rb,ra) as
    d ← RegRead(rd, 64)
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 64)
    endcase
    case op of
        A.MUX:
            a ← (c and d) or (b and not d)
    endcase
    RegWrite(ra, 64, a)
enddef
```

Exceptions

none

FIG. 68C

Operation codes

B	Branch

FIG. 69A

Format

B rd

31 24	23 18	17 12	11 6	5 0
B.MINOR	rd	0	0	B
8	6	6	6	6

FIG. 69B

Definition

```
def Branch(rd,rc,rb) as
    if (rc ≠ 0) or (rb ≠ 0) then
        raise ReservedInstruction
    endif
    d ← RegRead(rd, 64)
    if ($d_{1..0}$) ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    ProgramCounter ← $d_{63..2}$ || $0^2$
    raise TakenBranch
enddef
```

Exceptions

Reserved Instruction
Access disallowed by virtual address

FIG. 69C

Operation codes

B.BACK	Branch back

FIG. 70A

Format

B.BACK

bback()

31 24	23 18	17 12	11 6	5 0
B.MINOR	0	0	0	B.BACK
8	6	6	6	6

FIG. 70B

Definition

```
def BranchBack(rd,rc,rb) as
    c ← RegRead(rc, 128)
    if (rd ≠ 0) or(rc ≠ 0) or (rb ≠ 0) then
        raise ReservedInstruction
    endif
    a ← LoadMemory(ExceptionBase,ExceptionBase+Thread*128,128,L)
    if PrivilegeLevel > $c_{1..0}$ then
        PrivilegeLevel ← $c_{1..0}$
    endif
    ProgramCounter ← $c_{63..2}$ || $0^2$
    ExceptionState ← 0
    RegWrite(rd,128,a)
    raise TakenBranchContinue
enddef
```

Exceptions

Reserved Instruction
Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 70C

Operation codes

B.BARRIER	Branch barrier

FIG. 71A

Format

B.BARRIER rd

bbarrier(rd)

31 24	23 18	17 12	11 6	5 0
B.MINOR	rd	0	0	B.BARRIER
8	6	6	6	6

FIG. 71B

Definition

```
def BranchBarrier(rd,rc,rb) as
    if (rc $\neq$ 0) or (rb $\neq$ 0) then
        raise ReservedInstruction
    endif
    d $\leftarrow$ RegRead(rd, 64)
    if ($d_{1..0}$) $\neq$ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    ProgramCounter $\leftarrow d_{63..2} \| 0^2$
    FetchBarrier()
    raise TakenBranch
enddef
```

Exceptions

Reserved Instruction

FIG. 71C

Operation codes

B.AND.E	Branch and equal zero
B.AND.NE	Branch and not equal zero
B.E	Branch equal
B.GE	Branch greater equal signed
B.L	Branch signed less
B.NE	Branch not equal
B.GE.U	Branch greater equal unsigned
B.L.U	Branch less unsigned

Equivalencies

B.E.Z	Branch equal zero
B.G.Z[1]	Branch greater zero signed
B.GE.Z[2]	Branch greater equal zero signed
B.L.Z[3]	Branch less zero signed
B.LE.Z[4]	Branch less equal zero signed
B.NE.Z	Branch not equal zero
B.LE	Branch less equal signed
B.G	Branch greater signed
B.LE.U	Branch less equal unsigned
B.G.U	Branch greater unsigned
B.NOP	Branch no operation

B.E.Z rc,target	←	B.AND.E rc,rc,target
B.G.Z rc,target	⇐	B.L.U rc,rc,target
B.GE.Z rc,target	⇐	B.GE rc,rc,target
B.L.Z rc,target	⇐	B.L rc,rc,target
B.LE.Z rc,target	⇐	B.GE.U rc,rc,target
B.NE.Z rc,target	←	B.AND.NE rc,rc,target
B.LE rc,rd,target	→	B.GE rd,rc,target
B.G rc,rd,target	→	B.L rd,rc,target
B.LE.U rc,rd,target	→	B.GE.U rd,rc,target
B.G.U rc,rd,target	→	B.L.U rd,rc,target
B.NOP	←	B.NE r0,r0,$

Redundancies

B.E rc,rc,target	⇔	B.I target
B.NE rc,rc,target	⇔	*B.NOP*

FIG. 72A

Selection

class	op	compare	type
arithmetic		L GE *G* *LE*	NONE U
vs. zero		*L* *GE* *G* *LE* *E* *NE*	*Z*
bitwise	none AND	E NE	

Format

op rd,rc,target

if (op(rd,rc)) goto target;

31 24	23 18	17 12	11 0
op	rd	rc	offset
8	6	6	12

FIG. 72B

Definition

```
def BranchConditionally(op,rd,rc,offset) as
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    case op of
        B.E:
            $a \leftarrow d = c$
        B.NE:
            $a \leftarrow d \neq c$
        B.AND.E:
            $a \leftarrow (d \text{ and } c) = 0$
        BAND.NE:
            $a \leftarrow (d \text{ and } c) \neq 0$
        B.L:
            $a \leftarrow (rd = rc)\ ?\ (c < 0) : (d < c)$
        B.GE:
            $a \leftarrow (rd = rc)\ ?\ (c \geq 0) : (d \geq c)$
        B.L.U:
            $a \leftarrow (rd = rc)\ ?\ (c > 0) : ((0 \| d) < (0 \| c))$
        B.GE.U:
            $a \leftarrow (rd = rc)\ ?\ (c \leq 0) : ((0 \| d) \geq (0 \| c))$
    endcase
    if a then
        $\text{ProgramCounter} \leftarrow \text{ProgramCounter} + (\text{offset}_{11}^{50} \| \text{offset} \| 0^2)$
        raise TakenBranch
    endif
enddef
```

Exceptions

none

FIG. 72C

Operation codes

B.E.F. 16	Branch equal floating-point half
B.E.F. 32	Branch equal floating-point single
B.E.F. 64	Branch equal floating-point double
B.E.F.128	Branch equal floating-point quad
B.GE.F. 16	Branch greater equal floating-point half
B.GE.F. 32	Branch greater equal floating-point single
B.GE.F. 64	Branch greater equal floating-point double
B.GE.F.128	Branch greater equal floating-point quad
B.L.F. 16	Branch less floating-point half
B.L.F. 32	Branch less floating-point single
B.L.F. 64	Branch less floating-point double
B.L.F.128	Branch less floating-point quad
B.LG.F. 16	Branch less greater floating-point half
B.LG.F. 32	Branch less greater floating-point single
B.LG.F. 64	Branch less greater floating-point double
B.LG.F.128	Branch less greater floating-point quad

Equivalencies

B.LE.F. 16	Branch less equal floating-point half
B.LE.F. 32	Branch less equal floating-point single
B.LE.F. 64	Branch less equal floating-point double
B.LE.F.128	Branch less equal floating-point quad
B.G.F. 16	Branch greater floating-point half
B.G.F. 32	Branch greater floating-point single
B.G.F. 64	Branch greater floating-point double
B.G.F.128	Branch greater floating-point quad

B.LE.F.size rc,rd,target	→	B.GE.F.size rd,rc,target
B.G.F.size rc,rd,target	→	B.L.F.size rd,rc,target

FIG. 73A

Selection

number format	type	compare	size
floating-point	F	E LG L GE *G* *LE*	16 32 64 128

Format

op rd,rc,target

if (op(rd,rc)) goto target;

31 24	23 18	17 12	11 0
op	rd	rc	offset
8	6	6	12

FIG. 73B

Definition

```
def BranchConditional(FloatingPointop,rd,rc,offset) as
    case op of
        B.E.F.16, B.LG.F.16, B.L.F.16, B.GE.F.16:
            size ← 16
        B.E.F.32, B.LG.F.32, B.L.F.32, B.GE.F.32:
            size ← 32
        B.E.F.64, B.LG.F.64, B.L.F.64, B.GE.F.64:
            size ← 64
        B.E.F.128, B.LG.F.128, B.L.F.128, B.GE.F.128:
            size ← 128
    endcase
    d ← F(size,RegRead(rd, 128))
    c ← F(size,RegRead(rc, 128))
    v ← fcom(d, c)
    case op of
        BEF16, BEF32, BEF64, BEF128:
            a ← (v = E)
        BLGF16, BLGF32, BLGF64, BLGF128:
            a ← (v = L) or (v = G)
        BLF16, BLF32, BLF64, BLF128:
            a ← (v = L)
        BGEF16, BGEF32, BGEF64, BGEF128:
            a ← (v = G) or (v = E)
    endcase
    if a then
        ProgramCounter ← ProgramCounter + ($\text{offset}_{11}^{50}$ || offset || $0^2$)
        raise TakenBranch
    endif
enddef
```

Exceptions

none

FIG. 73C

Operation codes

B.I.F. 32	Branch invisible floating-point single
B.NI.F. 32	Branch not invisible floating-point single
B.NV.F. 32	Branch not visible floating-point single
B.V.F. 32	Branch visible floating-point single

FIG. 74A

Selection

number format	type	compare	size
floating-point	F	I NI NV V	32

Format

op rc,rd,target

if (op(rc,rd)) goto target;

31 24	23 18	17 12	11 0
op	rd	rc	offset
8	6	6	12

FIG. 74B

Definition

```
def n(a) as (a.t=QNAN) or (a.t=SNAN) enddef

def less(a,b) as fcom(a,b)=L enddef

def trxya,b,c,d) as (fcom(fabs(a),b)=G) and (fcom(fabs(c),d)=G) and (a.s=c.s) enddef


def BranchConditionalVisibilityFloatingPoint(op,rd,rc,offset) as
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    dx ← F(32,$d_{31..0}$)
    cx ← F(32,$c_{31..0}$)
    dy ← F(32,$d_{63..32}$)
    cy ← F(32,$c_{63..32}$)
    dz ← F(32,$d_{95..64}$)
    cz ← F(32,$c_{95..64}$)
    dw ← F(32,$d_{127..96}$)
    cw ← F(32,$c_{127..96}$)
    fl ← F(32,0x7f000000) // floating-point 1.0
    if (n(dx) or n(dy) or n(dz) or n(dw) or n(cx) or n(cy) or n(cz) or n(cw)) then
        a ← false
    else
        dv ← less(fabs(dx),dz) and less(fabs(dy),dz) and less(dz,f1) and (dz.s=0)
        cv ← less(fabs(cx),cz) and less(fabs(cy),cz) and less(cz,f1) and (cz.s=0)
        trz ← (less(f1,dz) and less(f1,cz)) or ((dz.s=1 and cz.s=1))
        tr ← trxy(dx,dz,cx,cz) or trxy(dy,dz,cy,cz) or trz
        case op of
            B.I.F.32:
                a ← tr
            B.NI.F.32:
                a ← not tr
            B.NV.F.32:
                a ← not (dv and cv)
            B.V.F.32:
                a ← dv and cv
        endcase
    endif
    if a then
        ProgramCounter ← ProgramCounter + (offset$_{11}^{50}$ || offset || $0^2$)
```

FIG. 74C

```
            raise TakenBranch
      endif
enddef
```

Exceptions

none

FIG. 74C *continued*

Operation codes

B.DOWN	Branch down

FIG. 75A

Format

B.DOWN rd

bdown(rd)

31 24	23 18	17 12	11 6	5 0
B.MINOR	rd	0	0	B.DOWN
8	6	6	6	6

FIG. 75B

Definition

```
def BranchDown(rd,rc,rb) as
    if (rc ≠ 0) or (rb ≠ 0) then
        raise ReservedInstruction
    endif
    d ← RegRead(rd, 64)
    if PrivilegeLevel > $d_{1..0}$ then
        PrivilegeLevel ← $d_{1..0}$
    endif
    ProgramCounter ← $d_{63..2} \| 0^2$
    raise TakenBranch
enddef
```

Exceptions

Reserved Instruction

FIG. 75C

Operation codes

B.GATE	Branch gateway

Equivalencies

B.GATE	← B.GATE 0

FIG. 76A

Format

B.GATE rb

bgate(rb)

31 24	23 18	17 12	11 6	5 0
B.MINOR	0	1	rb	B.GATE
8	6	6	6	6

FIG. 76B

Definition

```
def BranchGateway(rd,rc,rb) as
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 64)
    if (rd ≠ 0) or (rc ≠ 1) then
        raise ReservedInstruction
    endif
    if $c_{2..0} \neq 0$ then
        raise AccessDisallowedByVirtualAddress
    endif
    d ← $\text{ProgramCounter}_{63..2}+1$ || PrivilegeLevel
    if PrivilegeLevel < $b_{1..0}$ then
        m ← LoadMemoryG(c,c,64,L)
        if b ≠ m then
            raise GatewayDisallowed
        endif
        PrivilegeLevel ← $b_{1..0}$
    endif
    ProgramCounter ← $b_{63..2} \| 0^{2}$
    RegWrite(rd, 64, d)
    raise TakenBranch
enddef
```

Exceptions

Reserved Instruction
Gateway disallowed
Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 76C

Operation codes

B.HALT	Branch halt

FIG. 77A

Format

B.HALT

bhalt()

31 24	23 18	17 12	11 6	5 0
B.MINOR	0	0	0	B.HALT
8	6	6	6	6

FIG. 77B

Definition

```
def BranchHalt(rd,rc,rb) as
    if $(rd \neq 0)$ or $(rc \neq 0)$ or $(rb \neq 0)$ then
        raise ReservedInstruction
    endif
    FetchHalt()
enddef
```

Exceptions

Reserved Instruction

FIG. 77C

Operation codes

B.HINT	Branch Hint

FIG. 78A

Format

B.HINT badd,count,rd

bhint(badd,count,rd)

31 24	23 18	17 12	11 6	5 0
B.MINOR	rd	count	simm	B.HINT
8	6	6	6	6

simm ← badd-pc-4

FIG. 78B

Definition

```
def BranchHint(rd,count,simm) as
    d ← RegRead(rd, 64)
    if ($d_{1..0}$) ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    FetchHint(ProgramCounter +4 + (0 || simm || $0^2$), $d_{63..2}$ || $0^2$, count)
enddef
```

Exceptions

Access disallowed by virtual address

FIG. 78C

Operation codes

B.HINT.I	Branch Hint Immediate

FIG. 79A

Format

B.HINT.I badd,count,target

bhinti(badd,count,target)

31 24	23 18	17 12	11 0
B.HINT.I	simm	count	offset
8	6	6	12

simm ← badd-pc-4

FIG. 79B

Definition

```
def BranchHintImmediate(simm,count,offset) as
    BranchHint(ProgramCounter + 4 + (0 || simm || $0^2$), count,
            ProgramCounter + ($\text{offset}_{17}^{44}$ || offset || $0^2$))
enddef
```

Exceptions

none

FIG. 79C

Operation codes

B.I	Branch immediate

Redundancies

B.I target	⇔	B.E rc,rc,target

FIG. 80A

Format

B.I target

bi(target)

31 24	23 0
B.I	offset
8	24

FIG. 80B

Definition

```
def BranchImmediate(offset) as
    ProgramCounter ← ProgramCounter + $(\text{offset}_{23}^{38} \,\|\, \text{offset} \,\|\, 0^{2})$
    raise TakenBranch
enddef
```

Exceptions

none

FIG. 80C

Operation codes

B.LINK.I	Branch immediate link

FIG. 81A

Format

B.LINK.I target

blinki(target)

31 24	23 0
B.LINK.I	offset
8	24

FIG. 81B

Definition

```
def BranchImmediateLink(offset) as
    RegWrite(0, 64, ProgramCounter + 4)
    ProgramCounter ← ProgramCounter + ($offset_{23}^{38}$ || offset || $0^2$)
    raise TakenBranch
enddef
```

Exceptions

none

FIG. 81C

Operation codes

B.LINK	Branch link

Equivalencies

B.LINK	←	B.LINK 0=0
B.LINK rc	←	B.LINK 0=rc

FIG. 82A

Format

B.LINK rd=rc

31 24	23 18	17 12	11 6	5 0
B.MINOR	rd	rc	0	B.LINK
8	6	6	6	6

rb ← 0

FIG. 82B

Definition

```
def BranchLink(rd,rc,rb) as
    if rb ≠ 0 then
        raise ReservedInstruction
    endif
    c ← RegRead(rc, 64)
    if (c and 3) ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    RegWrite(rd, 64, ProgramCounter + 4)
    ProgramCounter ← $c_{63..2} \,||\, 0^2$
    raise TakenBranch
enddef
```

Exceptions

Reserved Instruction
Access disallowed by virtual address

FIG. 82C

Operation codes

S.D.C.S.64.A.B	Store double compare swap octlet aligned big-endian
S.D.C.S.64.A.L	Store double compare swap octlet aligned little-endian

FIG. 83A

Format

op rd@rc,rb

rd=op(rd,rc,rb)

31 24	23 18	17 12	11 6	5 0
S.MINOR	rd	rc	rb	op
8	6	6	6	6

FIG. 83B

Definition

```
def StoreDoubleCompareSwap(op,rd,rc,rb) as
    size ← 64
    lsize ← log(size)
    case op of
        SDCS64AL:
            order ← L
        SDCS64AB:
            order ← B
    endcase
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    d ← RegRead(rd, 128)
    if ($c_{2..0}$ ≠ 0) or ($b_{2..0}$ ≠ 0) then
        raise AccessDisallowedByVirtualAddress
    endif
    lock
        a ← LoadMemoryW($c_{63..0}$,$c_{63..0}$,64,order) || LoadMemoryW($b_{63..0}$,$b_{63..0}$,64,order)
        if (($c_{127..64}$ || $b_{127..64}$.) = a) then
            StoreMemory(($c_{63..0}$,$c_{63..0}$,64,order,$d_{127..64}$)
            StoreMemory($b_{63..0}$,$b_{63..0}$,64,order,$d_{63..0}$)
        endif
    endlock
    RegWrite(rd, 128, a)
enddef
```

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 83C

Operation codes

S.A.S.I.64.A.B	Store add swap immediate octlet aligned big-endian
S.A.S.I.64.A.L	Store add swap immediate octlet aligned little-endian
S.C.S.I.64.A.B	Store compare swap immediate octlet aligned big-endian
S.C.S.I.64.A.L	Store compare swap immediate octlet aligned little-endian
S.M.S.I.64.A.B	Store multiplex swap immediate octlet aligned big-endian
S.M.S.I.64.A.L	Store multiplex swap immediate octlet aligned little-endian

FIG. 84A

Selection

number format	op	size	alignment	ordering
add-swap	AS	64	A	L B
compare-swap	CS	64	A	L B
multiplex-swap	MS	64	A	L B

Format

S.op.I.64.align.order rd@rc,offset

rd=sopi64alignorder(rd,rc,offset)

31 24	23 18	17 12	11 0
op	rd	rc	offset
8	6	6	12

FIG. 84B

Definition

```
def StoreImmediateInplace(op,rd,rc,offset) as
    sizE ← 64
    lsize ← log(size)
    case op of
        SASI64AL, SCSI64AL, SMSI64AL:
            order ← L
        SASI64AB, SCSI64AB, SMSI64AB:
            order ← B
    endcase
    c ← RegRead(rc, 64)
    VirtAddr ← c + (offset$\}\}^{-lsize}$ || offset || $0^{lsize-3}$)
    if ($c_{lsize-4..0}$ ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    d ← RegRead(rd, 128)
    case op of
        SASI64AB, SASI64AL:
            lock
                a ← LoadMemoryW(c,VirtAddr,size,order)
                StoreMemory(c,VirtAddr,size,order,$d_{63..0}$+a)
            endlock
        SCSI64AB, SCSI64AL:
            lock
                a ← LoadMemoryW(c,VirtAddr,size,order)
                if (a = $d_{63..0}$) then
                    StoreMemory(c,VirtAddr,size,order,$d_{127..64}$)
                endif
            endlock
        SMSI64AB, SMSI64AL:
            lock
                a ← LoadMemoryW(c,VirtAddr,size,order)
                m ← ($d_{127..64}$ & $d_{63..0}$) | (a & ~$d_{63..0}$)
                StoreMemory(c,VirtAddr,size,order,m)
            endlock
    endcase
    RegWrite(rd, 64, a)
enddef
```

FIG. 84C

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 84C *continued*

Operation codes

S.A.S.64.A.B	Store add swap octlet aligned big-endian
S.A.S.64.A.L	Store add swap octlet aligned little-endian
S.C.S.64.A.B	Store compare swap octlet aligned big-endian
S.C.S.64.A.L	Store compare swap octlet aligned little-endian
S.M.S.64.A.B	Store multiplex swap octlet aligned big-endian
S.M.S.64.A.L	Store multiplex swap octlet aligned little-endian

FIG. 85A

Selection

number format	op	size	alignment	ordering
add-swap	A.S	64	A	L B
compare-swap	C.S	64	A	L B
multiplex-swap	M.S	64	A	L B

Format

op rd@rc,rb

rd=op(rd,rc,rb)

31 24	23 18	17 12	11 6	5 0
S.MINOR	rd	rc	rb	op
8	6	6	6	6

FIG. 85B

Definition

```
def StoreInplace(op,rd,rc,rb) as
    size ← 64
    lsize ← log(size)
    case op of
        SAS64AL, SCS64AL, SMS64AL:
            order ← L
        SAS64AB, SCS64AB, SMS64AB:
            order ← B
    endcase
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 64)
    VirtAddr ← c + ($b_{66-lsize..0}$ || $0^{lsize-3}$)
    if ($c_{lsize-4..0}$ ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    d ← RegRead(rd, 128)
    case op of
        SAS64AB, SAS64AL:
            lock
                a ← LoadMemoryW(c,VirtAddr,size,order)
                StoreMemory(c,VirtAddr,size,order,$d_{63..0}$+a)
            endlock
        SCS64AB, SCS64AL:
            lock
                a ← LoadMemoryW(c,VirtAddr,size,order)
                if (a = $d_{63..0}$) then
                    StoreMemory(c,VirtAddr,size,order,$d_{127..64}$)
                endif
            endlock
        SMS64AB, SMS64AL:
            lock
                a ← LoadMemoryW(c,VirtAddr,size,order)
                m ← ($d_{127..64}$ & $d_{63..0}$) | (a & ~$d_{63..0}$)
                StoreMemory(c,VirtAddr,size,order,m)
            endlock
    endcase
    RegWrite(rd, 64, a)
enddef
```

FIG. 85C

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 85C *continued*